Programowanie obiektowe (ang. *object-oriented programming*, OOP) – paradygmat programowania, w którym programy definiuje się za pomocą obiektów – elementów łączących *stan* (czyli dane, nazywane najczęściej atrybutami) i *zachowanie* (czyli procedury, tu: metody). Obiektowy program komputerowy wyrażony jest jako zbiór takich obiektów, komunikujących się pomiędzy sobą w celu wykonywania zadań.

Podejście to różni się od tradycyjnego programowania proceduralnego, gdzie dane i procedury nie są ze sobą bezpośrednio związane. Programowanie obiektowe ma ułatwić pisanie, konserwację i wielokrotne użycie programów lub ich fragmentów.

Cechy

Programowanie obiektowe wykorzystuje obiekty, ale nie wszystkie powiązane techniki i struktury są obsługiwane bezpośrednio w językach, które twierdzą, że obsługują OOP. Cechy wymienione poniżej są jednak powszechne wśród języków uważanych za silnie zorientowane na klasę i obiekt (lub wielo-paradygmatowe z obsługą OOP)[1][2][3][4].

Obiekty i klasy

Języki, które wspierają programowanie obiektowe zwykle używają dziedziczenia dla ponownego użycia kodu oraz rozciągliwości w formie klas lub prototypów. Te, które używają klas opierają się na dwóch głównych konceptach:

- Klasy – definicje formatu danych oraz dostępnych procedur dla danego typu lub klasy obiektu, mogą same także zawierać dane oraz procedury (znane jako metody klas), na przykład, klasy zawierają dane oraz metody.
- Obiekty – instancje klas.

Obiekty czasem odpowiadają rzeczom występującym w realnym świecie. Dla przykładu, program graficzny może zawierać obiekty takie jak *koło*, *kwadrat*, *menu*. System sklepu internetowego może zawierać obiekty typu *koszyk, klient* czy *produkt*[5]. Czasem obiekty reprezentują bardziej abstrakcyjne jednostki, takie jak obiekt reprezentujący otwarty plik lub obiekt, który zapewnia serwerowi konwertującemu jednostki miar z amerykańskich na metryczne.

Jak pisze Junade Ali w Mastering PHP Design Patterns:

„Programowanie obiektowe to więcej niż klasy i obiekty; to cały paradygmat programowania bazujący na obiektach (strukturach danych), które zawierają obszary danych (pola) oraz metody. Zrozumienie tego jest bardzo ważne; używanie klas do zorganizowania zbioru niezwiązanych ze sobą metod nie jest podejściem obiektowym.”[6]

Każdy obiekt jest instancją konkretnej klasy; na przykład obiekt, którego obszar nazwy to „Mary”, może być instancją klasy „Pracownicy”. Procedury w programowaniu obiektowym nazywane są metodami, a zmienne obszarami, członkami, atrybutami albo właściwościami. Sprowadza się to do następujących określeń:

- Zmienne klasy – należą do całości klasy, jest tylko jedna kopia każdej

- Instancje zmiennych lub atrybutów – dane, które należą do indywidualnego obiektu, każdy obiekt ma ich własną kopię
- Zmienne składowe – odwołują się zarówno do klas oraz zmiennych instancji, które są zdefiniowane przez konkretne klasy
- Metody w klasach – należą do całości klas i mają dostęp tylko do zmiennych klas oraz wprowadzanych w wywoływaniu procedur
- Metody instancji – należą do indywidualnych obiektów, mają dostęp do zmiennych instancji dla specyficznych obiektów, dla których są wywoływane, wprowadzanych oraz zmiennych klas

Dostęp do obiektów wygląda podobnie jak do zmiennych z kompleksową wewnętrzną strukturą, w wielu językach wskaźniki działają dobrze, będąc naturalną referencją do pojedynczej instancji podanego obiektu w pamięci za pomocą kopca albo stosu. Zapewniają warstwę abstrakcji, która może być użyta do odseparowania wewnętrznego kodu od zewnętrznego. Zewnętrzny kod może użyć obiektu poprzez wywołanie specyficznej metody instancji z konkretnym zestawem wprowadzanych parametrów, czytanych jako zmienna instancji, lub pisanie do zmiennej instancji. Obiekty są wywoływane przez specjalne typy metod w klasie znanych jako konstruktory. Program może stworzyć wiele instancji tej samej klasy w czasie działania, które działają niezależnie od siebie. W ten łatwy sposób procedury mogą być użyte na różnych zestawach danych.

Programowanie obiektowe używające klas jest czasem nazywane programowaniem klasowym, podczas gdy prototypowe programowanie zwykle nie używa klas. W wyniku tego, znacząco różna, aczkolwiek analogiczna terminologia jest używana do definicji obiektu oraz instancji.

W niektórych językach klasy i obiekty mogą być zaspokojone używając innych konceptów jak Cechy i Domieszki (programowanie obiektowe).

Programowanie klasowe kontra prototypowe

W językach bazujących na klasach są one definiowane na początku, a obiekty tworzone są w oparciu o klasy. Jeżeli dwa obiekty „jabłko” oraz „pomarańcza” są utworzone w klasie „Owoce”, są w wyniku dziedziczenia owocami i możemy być pewni, że można się nimi posługiwać w ten sam sposób, dla przykładu programista może oczekiwać istnienie tych samych atrybutów takich jak „kolor”, „zawartość cukru” lub „czy jest dojrzały”. W językach prototypowych obiekty są pierwotnymi bytami. Żadne klasy nawet nie istnieją. Prototyp obiektu jest po prostu kolejnym obiektem, do którego podłączony jest inny. Każdy obiekt ma jeden „link prototypowy” (i tylko jeden). Nowe obiekty mogą być stworzone bazując na tych, które już istnieją i są wybrane jako ich prototyp. Możesz nazwać „jabłko” i „pomarańczę” dwoma różnymi obiektami, jeżeli istnieje obiekt „owoc”, a zarówno „jabłko”, jak i „pomarańcza” mają „owoc” za prototyp. Idea klasy „owoc” nie istnieje wyraźnie, ale klasa równoważna obiektu dzieli ten sam prototyp. Atrybuty i metody prototypów są delegowane do wszystkich obiektów równoważnych klas zdefiniowanych przez ich prototyp. Atrybuty i metody posiadane indywidualnie przez obiekt nie mogą być współdzielone z innymi obiektami o tej samej klasie równoważnej, dla przykładu atrybut „zawartość cukru” może być niespodziewanie nieobecny w „jabłku”. Tylko pojedyncze dziedziczenie może być zaimplementowane poprzez prototyp.

Podstawowe założenia paradygmatu obiektowego

Istnieje pewna różnica zdań co do tego, jakie cechy języków programowania czynią je obiektowymi. Powszechnie uważa się, że najważniejsze są następujące cechy:

Abstrakcja

Każdy obiekt w systemie służy jako model abstrakcyjnego „wykonawcy", który może wykonywać pracę, opisywać i zmieniać swój stan oraz komunikować się z innymi obiektami w systemie bez ujawniania, w jaki sposób zaimplementowano dane cechy. Procesy, funkcje lub metody mogą być również abstrahowane, a kiedy tak się dzieje, konieczne są rozmaite techniki rozszerzania abstrakcji.

Hermetyzacja

Czyli ukrywanie implementacji, enkapsulacja. Zapewnia, że obiekt nie może zmieniać stanu wewnętrznego innych obiektów w nieoczekiwany sposób. Tylko własne metody obiektu są uprawnione do zmiany jego stanu. Każdy typ obiektu prezentuje innym obiektom swój interfejs, który określa dopuszczalne metody współpracy. Pewne języki osłabiają to założenie, dopuszczając pewien poziom bezpośredniego (kontrolowanego) dostępu do „wnętrzności" obiektu. Ograniczają w ten sposób poziom abstrakcji. Przykładowo w niektórych kompilatorach języka C++ istnieje możliwość tymczasowego wyłączenia mechanizmu enkapsulacji; otwiera to dostęp do wszystkich pól i metod prywatnych, ułatwiając programistom pracę nad pośrednimi etapami tworzenia kodu i znajdowaniem błędów.

Polimorfizm

Referencje i kolekcje obiektów mogą dotyczyć obiektów różnego typu, a wywołanie metody dla referencji spowoduje zachowanie odpowiednie dla pełnego typu obiektu wywoływanego. Jeśli dzieje się to w czasie działania programu, to nazywa się to *późnym wiązaniem* lub *wiązaniem dynamicznym*. Niektóre języki udostępniają bardziej statyczne (w trakcie kompilacji) rozwiązania polimorfizmu – na przykład szablony i przeciążanie operatorów w C++.

Dziedziczenie

Porządkuje i wspomaga polimorfizm i enkapsulację dzięki umożliwieniu definiowania i tworzenia specjalizowanych obiektów na podstawie bardziej ogólnych. Dla obiektów specjalizowanych nie trzeba redefiniować całej funkcjonalności, lecz tylko tę, której nie ma obiekt ogólniejszy. W typowym przypadku powstają grupy obiektów zwane *klasami*, oraz grupy klas zwane *drzewami*. Odzwierciedlają one wspólne cechy obiektów.

Historia programowania obiektowego

Koncepcja programowania obiektowego pierwotnie pojawiła się w Simuli 67, języku zaprojektowanym do zastosowań symulacyjnych, stworzonym przez Ole-Johana Dahla i Kristena Nygaarda z Norsk Regnesentral w Oslo. (Mówi się, że pracowali oni nad symulacjami zachowania się statków i mieli kłopoty z opanowaniem wszystkich zależności, jakie wywierały na siebie nawzajem wszystkie parametry statków w symulacji. Wtedy wpadli na pomysł, by pogrupować typy statków w różne klasy obiektów, a każda z klas *sama* odpowiadałaby za określanie własnych danych i zachowań.) Później koncepcja została

dopracowana w języku Smalltalk, stworzonym w Simuli w Xerox PARC, ale zaprojektowanym jako w pełni dynamiczny system, w którym obiekty mogą być tworzone i modyfikowane „w locie", a nie system oparty na statycznych programach.

Programowanie obiektowe zyskało status techniki dominującej w połowie lat 80., głównie ze względu na wpływ C++, stanowiącego rozszerzenie języka C. Dominacja C++ została utrwalona przez wzrost popularności graficznych interfejsów użytkownika, do tworzenia których programowanie obiektowe nadaje się szczególnie dobrze.

W tym okresie cechy obiektowe dodano do wielu języków programowania, w tym Ady, BASIC-a, Lispu, Pascala i innych. Dodanie obiektowości do języków, które pierwotnie nie były do niej przystosowane, zrodziło szereg problemów z kompatybilnością i konserwacją kodu. Z kolei „czysto" obiektowym językom brakowało cech, z których programiści przyzwyczajeni byli korzystać. By zapełnić tę lukę, podejmowano liczne próby stworzenia języków obiektowych dostarczających jednocześnie „bezpiecznych" elementów proceduralności. Eiffel Bertranda Meyera był wczesnym przykładem dość udanego języka spełniającego te założenia; obecnie został on w zasadzie całkowicie zastąpiony przez Javę.

Podobnie, jak programowanie funkcyjne doprowadziło do udoskonalenia technik takich, jak programowanie strukturalne, do współczesnych metod projektowania oprogramowania obiektowego zaliczają się takie usprawnienia, jak *wzorce projektowe* (ang. *design patterns*), *design by contract* i języki modelowania (np. UML).

Obiektowość rozprzestrzeniła się dość znacznie, jednak zwykle w systemach hybrydowych, w połączeniu z programowaniem niskopoziomowym (asembler), funkcyjnym (OCaml, niektóre dialekty Lispu), sieciowym (Java), skryptowym (Perl, Python, Ruby) itd. Systemy czysto obiektowe jak Smalltalk nie znalazły zbyt szerokiego zastosowania.

Podział

Można wyróżnić dwa zasadnicze podtypy programowania obiektowego:

- Programowanie oparte na klasach – definiowane są klasy, czyli typy zmiennych, a następnie tworzone są obiekty, czyli zmienne (w uproszczeniu) tych typów.
- Programowanie oparte na prototypach – w tym podejściu nie istnieje pojęcie klasy. Nowe obiekty tworzy się w oparciu o istniejący już obiekt – prototyp, po którym dziedziczone są pola i metody i można go rozszerzać o nowe. Spotykany raczej w językach interpretowanych, np. JavaScript.

Elementarna charakterystyka popularnych języków obiektowych

Poniższa charakterystyka dopuszcza bardzo dużą różnorodność – i w rzeczywistości, o ile systemy programowania strukturalnego czy funkcyjnego były do siebie stosunkowo podobne, o tyle systemy obiektowe różnią się w dużym stopniu.

- Dziedziczenie wielokrotne:
  - jest: C++, Incr Tcl, Lisp, Perl, Python
  - tylko interfejsy: C#, Java, Objective-C, ObjectPascal, PHP5

    - tylko metody: Ruby
- Klasa jest obiektem:
    - tak: Java (za pomocą obiektu Class), JavaScript, Lisp, Objective-C, Python, Ruby,
    - nie: C++, C#, ObjectPascal, OCaml, PHP5
- Wszystkie obiekty wywodzą się z jednego korzenia i muszą mieć nadklasę:
    - tak: C#, Incr Tcl, Java, Lisp, Objective-C, ObjectPascal, Perl (klasa UNIVERSAL), Python (klasy nowego typu), Ruby
    - nie: C++, OCaml, PHP5, Python (klasy starego typu)
- Obiekt można pytać o to, do której podklasy należy:
    - tak: C++ (RTTI), C#, Incr Tcl, Java (RTTI lub mechanizm refleksji), JavaScript, Lisp, Objective-C, ObjectPascal (RTTI), Perl (metoda isa), PHP5 (mechanizm refleksji), Python, Ruby
    - nie: OCaml
- Typy generyczne (uogólnione):
    - tak: C# 2.0, Java 1.5, ObjectPascal (Delphi 2009, FreePascal 2.2)
    - tak (przez mechanizm szablonów): C++, D
    - nie dotyczy: PHP5, JavaScript
- Przeciążanie operatorów:
    - tak: C++, C#, Lisp, ObjectPascal (od BDS 2006), Perl, Python, Ruby
    - nie: Java (choć przeciążonych jest kilka operatorów wbudowanych, np. + w klasie String), JavaScript, Objective-C, ObjectPascal (do BDS 2006), OCaml, PHP5 (podobnie jak w Javie przeciążone są niektóre operatory – wywołania składowych i metod, od PHP 5.2 także operacje tablicowe)